DAB 303

Social Media Ad Performance Optimization

(Individual Project)

SUBMITTED TO: -

Vicky Tao

SUBMITTED BY: -

Manish Kataria (W0865937)

Content

Introduction

We are working on the "Social Media Ad Performance Optimization" project, which aims to improve how paid ads perform on social media platforms. As businesses increasingly depend on digital ads to connect with customers and drive sales, it's essential to make sure that ad performance is optimized to get the best returns on ad spending. In this project, we will use data-driven insights to enhance ad targeting, engagement, and conversion rates. We aim to refine ad strategies using advanced techniques like multivariate testing and reinforcement learning, to reach the right audience and boost profitability.

Project Overview: In this section, we introduce the idea of improving social media ad performance. We explain why it's important to optimize paid ads for better returns and more engagement with the right audience.[1]

Significance of Social Media Ad Performance Optimization: We highlight how businesses depend on social media ads to generate leads and sales. By improving ads, we make sure the ad budget is used effectively.[2]

Techniques Utilized: We mention the key techniques we're using, like **multivariate testing** and **reinforcement learning**, and explain why we chose them to meet our goals.[3]

Now, in today's digital world, we see that social media advertising is one of the most powerful ways for brands to connect with their target audience. However, with more competition and different audience behaviors, we also need to focus on improving how well these ads perform. Here, we explore how we can use machine learning and data-driven methods, i.e., cluster analysis, to make ads work better, reduce costs, and boost return on investment (ROI). Moreover, this approach helps us understand our audience more clearly and make smarter ad decisions.

Objective

- **Enhancing the Performance of Paid Advertisements**: The main goal of this project is to make paid ads work better on social media. We do this by analyzing data to fine-tune ad strategies, ensuring they bring in the most return.
- **Key Metrics**: We focus on two important metrics: **lead generation** (how many potential customers engage with the ads) and **cost per acquisition** (how much it costs to get one customer).

- **Expected Impact**: We expect to improve both metrics—getting more leads while reducing the cost to get them, making ads more cost-effective.

The main goal of this project is to optimize paid social media ads by:

Increasing Lead Generation: We aim to identify the best-performing ad elements and use them to attract more potential customers.

Reducing Cost Per Acquisition (CPA): We want to make sure that ad spending is efficient by focusing on strategies that generate conversions at a lower cost.

- Identifying patterns in how users engage with ads and convert.
- Predicting ROI using machine learning models.
- Segmenting ads and user responses with Cluster Analysis.
- Recommending targeted improvements based on data insights.

Techniques Employed

- **Multivariate Testing**: In this technique, we will test multiple versions of an ad at the same time to see how different combinations of elements like images, text, and call-to-action impact user engagement and conversion rates. By comparing the results, we will find the most effective ad combinations.
- **Reinforcement Learning**: This technique will help us continuously improve ad strategies based on real-time feedback. The model will learn from the ongoing performance of ads and adjust targeting to optimize conversions while minimizing costs. It will dynamically change ad content and audience targeting for better cost efficiency.
- **Choosing Models**: Also, we find a data to perform prediction using four different model that are, Ransom Forest classifier, Logistic regression, DNN and LSTM.
- We use **Cluster Analysis** with K-Means to group ads based on engagement and conversion patterns.

Data Sources

We will use the following data sources:

- **Social Media Engagement Metrics**: This includes likes, shares, comments, impressions, and click-through rates (CTR), which help us understand how users are interacting with the ads.
- **Ad Conversion Rates**: This data shows how many users take a desired action (like signing up or making a purchase) after interacting with an ad. It will help us measure how successful the ads are at generating re conversions.
- **Dataset**: We use Kaggle data set with name **Social Media Advertising Dataset.[4]**

EDA using Statistics

```
[1] import pandas as pd
    import numpy as np
    import seaborn as sns
    import matplotlib.pyplot as plt
    from sklearn.preprocessing import OneHotEncoder
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import mean_absolute_error, r2_score
    from sklearn.ensemble import RandomForestRegressor
    from sklearn.linear_model import LinearRegression
    import tensorflow as tf
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, LSTM

    # Load the dataset using Pandas
    df = pd.read_csv("Social_Media_Advertising.csv")
```

```
[5] df.head()
```

	Campaign_ID	Target_Audience	Campaign_Goal	Duration	Channel_Used	Conversion_Rate	Acquisition_Cost	ROI	Location	Language	Clicks	Impressions	Engagement_Score	Customer_Segment	Date	Company
0	529013	Men 35-44	Product Launch	15 Days	Instagram	0.15	500.0	5.790000	Las Vegas	Spanish	500	3000	7	Health	2022-02-25	Aura Align
1	275352	Women 45-60	Market Expansion	15 Days	Facebook	0.01	500.0	7.210000	Los Angeles	French	500	3000	5	Home	2022-05-12	Hearth Harmony
2	692322	Men 45-60	Product Launch	15 Days	Instagram	0.08	500.0	0.430000	Austin	Spanish	500	3000	9	Technology	2022-06-19	Cyber Circuit
3	675757	Men 25-34	Increase Sales	15 Days	Pinterest	0.03	500.0	0.909824	Miami	Spanish	293	1937	1	Health	2022-09-08	Well Wish
4	535900	Men 45-60	Market Expansion	15 Days	Pinterest	0.13	500.0	1.422828	Austin	French	293	1937	1	Home	2022-08-24	Hearth Harmony

This image shows a Python code snippet running in a data analysis environment like Jupyter Notebook or Google Colab. The code is for loading and analyzing a dataset related to social media advertising.

What the Code Does:

1. Importing Libraries

We import several libraries to handle data and build machine learning models:

- pandas (pd) – For working with structured data (tables).
- NumPy (np) – For numerical operations.
- seaborn (sns) & matplotlib.pyplot (plt) – For data visualization.

- sklearn.preprocessing.OneHotEncoder – To convert categorical data into numbers.
- sklearn.model_selection.train_test_split – To split data into training and testing sets.
- sklearn.metrics.mean_absolute_error, r2_score – To evaluate model accuracy.
- sklearn.ensemble.RandomForestRegressor – For making predictions using a machine learning model.
- sklearn.linear_model.LinearRegression – Another model for predicting trends.
- tensorflow (tf) & keras – For deep learning models.
- tensorflow.keras.models.Sequential – To build neural networks.
- tensorflow.keras.layers.Dense, LSTM – Layers for deep learning models.

2. Loading the Dataset

We load a dataset named **"Social_Media_Advertising.csv"** into a table (Data Frame) using pandas:

3. Viewing the First Few Rows

This command shows the first 5 rows of the dataset to check if the data loaded correctly.

What the Dataset Includes:

The dataset has columns like:

- **Campaign_ID** – Unique ID for each ad campaign.
- **Target_Audience** – The group targeted (e.g., "Men 35-44").
- **Campaign_Goal** – Purpose of the campaign (e.g., "Increase Sales").
- **Duration** – Campaign length in days.
- **Channel_Used** – Social media platform (Facebook, Instagram, etc.).
- **Conversion_Rate** – How many people took action after seeing the ad.
- **Acquisition_Cost** – The cost of acquiring a customer.
- **ROI** – Return on Investment for the campaign.
- **Clicks, Impressions, Engagement_Score** – Metrics to measure ad performance.
- **Location & Language** – Targeted audience details.

- **Company** – The business running the campaign.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 16 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   Campaign_ID       300000 non-null  int64  
 1   Target_Audience   300000 non-null  object 
 2   Campaign_Goal     300000 non-null  object 
 3   Duration          300000 non-null  object 
 4   Channel_Used      300000 non-null  object 
 5   Conversion_Rate   300000 non-null  float64
 6   Acquisition_Cost  300000 non-null  float64
 7   ROI               300000 non-null  float64
 8   Location          300000 non-null  object 
 9   Language          300000 non-null  object 
 10  Clicks            300000 non-null  int64  
 11  Impressions       300000 non-null  int64  
 12  Engagement_Score  300000 non-null  int64  
 13  Customer_Segment  300000 non-null  object 
 14  Date              300000 non-null  object 
 15  Company           300000 non-null  object 
dtypes: float64(3), int64(4), object(9)
memory usage: 36.6+ MB
```

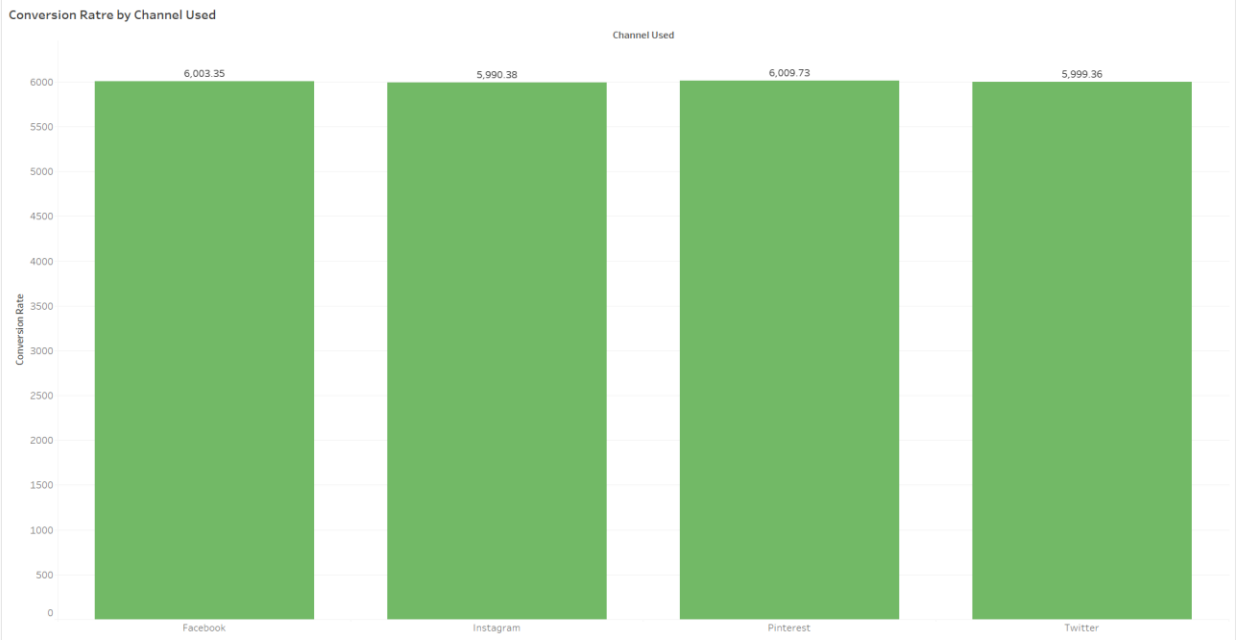
This image displays the result of the df.info() command, which summarizes the structure of a dataset named **df**, loaded from 'Social_Media_Advertising.csv'.

What This Means for Us:

- **Rows & Columns:** The dataset has **300,000 rows** and **16 columns**.
- **No Missing Values:** Every column contains **300,000 non-null entries**, meaning there are no missing values.
- **Data Types:**
    - **Integer (int64)** → Used for numerical values like **Campaign_ID, Clicks, Impressions, Engagement_Score**.
    - **Float (float64)** → Used for decimal values like **Conversion_Rate, Acquisition_Cost, ROI**.
    - **Object (String/Text)** → Includes **Target_Audience, Campaign_Goal, Duration, Channel_Used, Location, Language, Customer_Segment, Date, Company**.

EDA using Visualizations

Bar chart



Here, we have a bar chart titled "Conversion Rate by Channel Used." It shows the conversion rate for four social media channels: Facebook, Instagram, Pinterest, and Twitter.

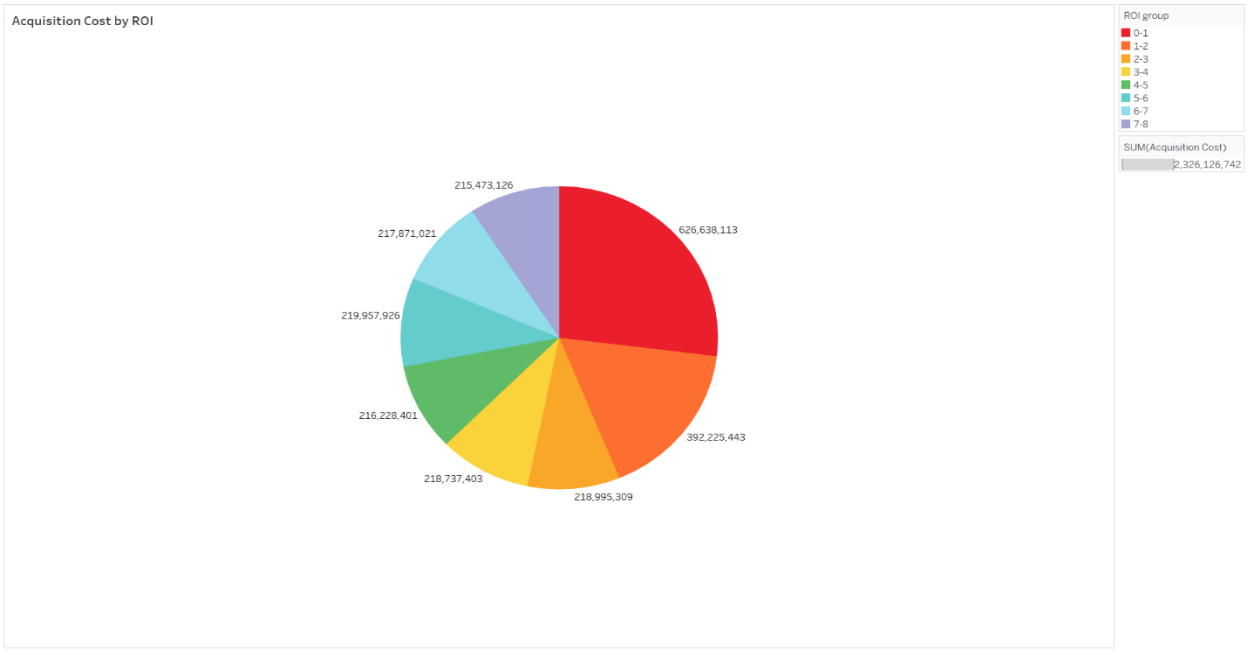
- **X-axis**: Represents the "Channel Used," i.e., Facebook, Instagram, Pinterest, and Twitter.
- **Y-axis**: Represents the "Conversion Rate," ranging from 0 to 6000.
- **Bars**: Each channel has one vertical bar. The height of the bar shows its conversion rate.

Here's what we can see:

- Facebook: ~6003.35
- Instagram: ~5930.38
- Pinterest: ~6059.73
- Twitter: ~5966.36

Moreover, Pinterest has the highest conversion rate, while Instagram has the lowest.

Pie Chart



Here, we have a pie chart titled **"Acquisition Cost by ROI"**, showing how acquisition costs are distributed across different ROI groups.

What the chart shows:

- **Title**: Clearly indicates the purpose of the chart.
- **Pie Slices**: The chart is divided into slices, each representing a different ROI group.
- **Color Coding**: Each ROI group has a unique color, as shown in the legend on the right:
    - **Red**: 0-1
    - **Orange**: 1-2
    - **Yellow**: 2-3
    - **Light Green**: 3-4
    - **Dark Green**: 4-5
    - **Light Blue**: 5-6
    - **Medium Blue**: 6-7

- **Light Purple**: 7-8

Key Observations:

- The **largest slice (red)** represents the **0-1 ROI group**, with the highest acquisition cost of **626,639,113**.
- The **second largest slice (orange)** corresponds to the **1-2 ROI group**, with an acquisition cost of **350,225,445**.
- The remaining slices represent lower acquisition costs as ROI increases.

Moreover, the total acquisition cost across all ROI groups is **2,326,126,742**, as shown on the right. This highlights that most acquisition costs are from activities with a lower ROI (between 0 and 2).

Tree Map

Here, we have a **treemap** titled **"Acquisition Cost by Target Audience,"** showing how acquisition costs are distributed among different audience segments.

What the treemap shows:

- **Rectangles**: Each rectangle represents a **target audience segment**.

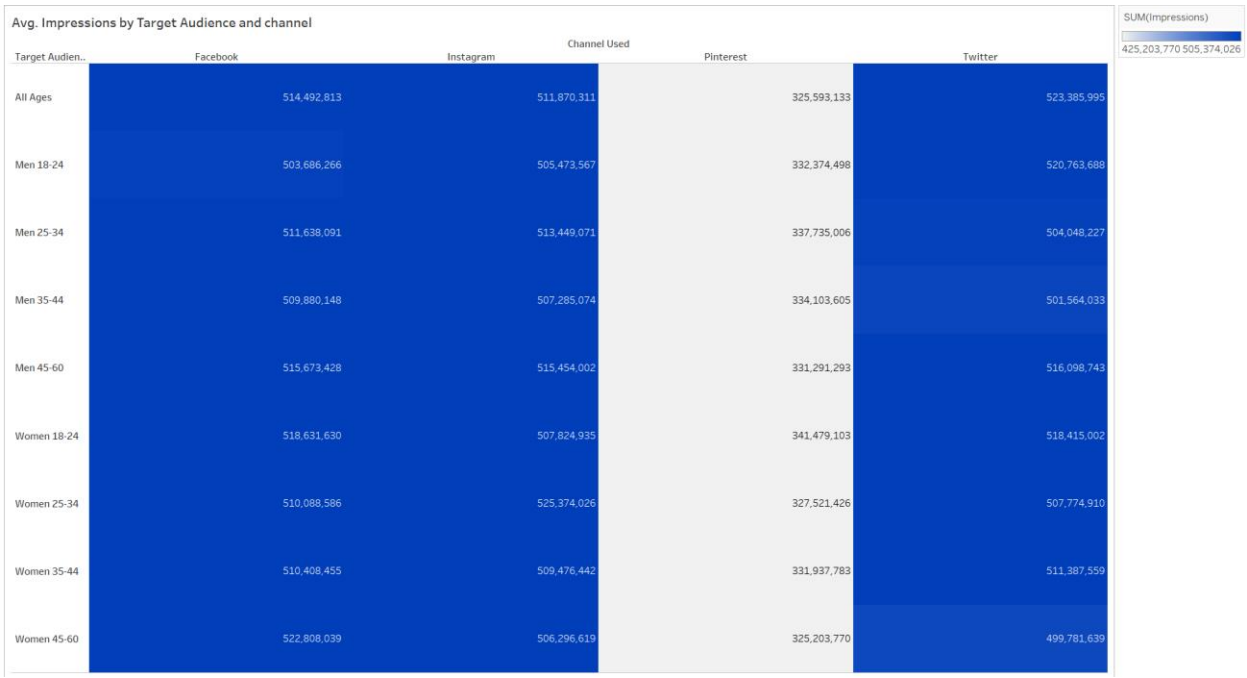
- **Size of Rectangles**: A larger rectangle means a **higher acquisition cost** for that audience.
- **Color Coding**:
    - **Darker shades (likely red)** → Higher acquisition costs
    - **Lighter shades (likely green/grey)** → Lower acquisition costs
- **Labels**: Each rectangle displays the **target audience name** and **acquisition cost**.

Key Observations:

- **Men 35-44 has the Highest Acquisition Cost**: The largest and darkest rectangle, with **298,721,539**.
- **Men 25-34 follows closely**: Also a large, dark-colored rectangle, with **298,728,806**.
- **Other Segments Have Lower Costs**:
    - **Women 18-24**: 261,445,417
    - **Men 45-60**: 258,780,750
    - **Women 25-34**: 258,513,578
    - **Women 35-44**: 257,984,532
    - **Women 45-60**: 258,243,023
    - **All Ages**: 258,969,404
- **Men 18-24 has a Moderate Cost**: **257,830,744**, with a medium-sized rectangle.

Moreover, this treemap clearly highlights that **Men 35-44 and Men 25-34 have the highest acquisition costs**, making them the most expensive segments in this dataset.

Heat Map



Here, we have a **heatmap** titled **"Avg. Impressions by Target Audience and Channel,"** showing the average impressions across different audience segments and social media platforms.

What the heatmap shows:

- **Rows**: Represent different **target audience segments**, i.e.,
    - All Ages
    - Men 18-24, 25-34, 35-44, 45-60
    - Women 18-24, 25-34, 35-44, 45-60
- **Columns**: Represent the **social media channels**, i.e., Facebook, Instagram, Pinterest, and Twitter.
- **Cells**: Show the **average number of impressions** for each audience-channel pair.
- **Color Intensity**: Darker blue shades indicate higher impressions, while lighter shades indicate lower impressions.
- **Legend**: On the right, it shows impression values, with the highest reaching **635,283,701 to 926,574,026**.

Key Observations:

- **Facebook & Instagram Lead**: These channels have the **highest average impressions**, with darker blue shades in most target audience segments.
- **Target Audience Differences**: Some segments, like **Men 45-60** and **Women 45-60**, see slightly higher impressions on Facebook.
- **Pinterest & Twitter Lower**: These channels show **fewer impressions** for most audience segments, as seen in the lighter blue shades.

Moreover, this heatmap helps us quickly compare which channels perform best for different target audiences, showing that **Facebook and Instagram dominate in impressions.**

Prediction Using models

```
[7]  df["Acquisition_Cost"] = df["Acquisition_Cost"].replace('[\$,]', '', regex=True).astype(float)
     df.dropna(inplace=True)

     # Descriptive Statistics
     numeric_columns = ["Conversion_Rate", "Acquisition_Cost", "ROI", "Clicks", "Impressions", "Engagement_Score"]
     numeric_stats = df[numeric_columns].describe()
```

```
[8]  roi_stats = {
         "Average_ROI": df["ROI"].mean(),
         "Max_ROI": df["ROI"].max(),
         "Min_ROI": df["ROI"].min()
     }

     channel_grouped = df.groupby("Channel_Used")["ROI"].agg(["mean", "max", "min"])
     goal_grouped = df.groupby("Campaign_Goal")["ROI"].agg(["mean", "max", "min"])
     audience_grouped = df.groupby("Target_Audience")["ROI"].agg(["mean", "max", "min"]).sort_values(by="mean")
     channel_target_stats = df.groupby(["Channel_Used", "Target_Audience"]).agg({
         "Conversion_Rate": "mean", "ROI": "mean"
     }).reset_index()
```

```
[9]  # Convert 'Date' column to datetime
     df["Date"] = pd.to_datetime(df["Date"])
```

This Python code performs several data cleaning, transformation, and descriptive statistics tasks on the df DataFrame, which holds the social media advertising data.

1. Cleaning the 'Acquisition_Cost' Column:

- **Cleaning:**
    - The replace() function removes the dollar signs ($) and commas (,) in the 'Acquisition_Cost' column. It uses regex to match these characters and replace them.

- After removing the non-numeric characters, the column is converted to a **float** type using astype(float) to allow numerical operations.

- **Handling Missing Values:**
  - The dropna() method removes rows containing missing values (NaN). This ensures that the data is complete before analysis.

2. Descriptive Statistics:

- **Description of Numeric Columns:**
  - A list of numeric columns is created, and describe() is applied to them. This generates key statistics such as:
    - **Count**: Number of non-null entries.
    - **Mean**: The average value.
    - **Std**: The standard deviation, indicating the spread of the data.
    - **Min, 25%, 50%, 75%, Max**: These provide a summary of the distribution, with percentiles giving insight into the data spread.

3. Calculating ROI Statistics and Grouped Aggregations:

- **ROI Stats:**
  - roi_stats calculates key statistics for the **ROI** column: average (mean), maximum (max), and minimum (min).

- **Grouped Aggregations:**
  - **Channel Grouping:** Groups by the **'Channel_Used'** column and calculates the **mean, max, and min** ROI for each channel (e.g., Facebook, Instagram).
  - **Goal Grouping:** Groups by **Campaign_Goal** and calculates ROI statistics for each goal.
  - **Audience Grouping:** Groups by **Target_Audience** and calculates ROI statistics, then sorts by the mean ROI.
  - **Channel and Audience Grouping:** Groups by both **Channel_Used** and **Target_Audience**, then calculates the **mean conversion rate and ROI** for each combination. The result is reset to a regular index after aggregation.

4. Converting the 'Date' Column to Datetime:

- **Datetime Conversion:**
    - The Date column is converted into a **datetime** format using pd.to_datetime(). This allows for time-based analysis and operations, such as filtering or aggregating by date.

This code performs several important tasks:

- **Data Cleaning:** It cleans the 'Acquisition_Cost' column by removing non-numeric characters and converts it to a numeric type.
- **Descriptive Statistics:** It calculates basic stats for numerical columns like **Conversion Rate, ROI, Clicks**, and others.
- **Aggregated Statistics:** It calculates the average, max, and min ROI for different segments (e.g., by **Channel_Used**, **Campaign_Goal**, and **Target_Audience**).
- **Date Conversion:** It converts the 'Date' column into a datetime object to facilitate time-based analysis.

These steps help prepare the data for further analysis or machine learning, providing insights into the performance of different social media advertising campaigns.

```
[11] # Data Cleaning and Preprocessing
     df["Acquisition_Cost"] = df["Acquisition_Cost"].replace('[\$,]', '', regex=True).astype(float)
     df.dropna(inplace=True)
     df["Date"] = pd.to_datetime(df["Date"])

     # Feature Selection & Encoding
     categorical_cols = ['Target_Audience', 'Campaign_Goal', 'Duration', 'Channel_Used', 'Location',
                         'Language', 'Customer_Segment', 'Company']
     encoder = OneHotEncoder(sparse_output=False, drop='first')
     categorical_encoded = encoder.fit_transform(df[categorical_cols])
     categorical_encoded_df = pd.DataFrame(categorical_encoded, columns=encoder.get_feature_names_out(categorical_cols))

     df_final = pd.concat([df.drop(columns=categorical_cols + ['Date']), categorical_encoded_df], axis=1)

     # Train-Test Split
     X = df_final.drop("ROI", axis=1)
     y = df_final["ROI"]
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
     # Function to Train & Evaluate Model
     def train_evaluate_model(model, X_train, y_train, X_test, y_test, model_name):
         model.fit(X_train, y_train)
         predictions = model.predict(X_test)
         print(predictions)
         mae = mean_absolute_error(y_test, predictions)
         r2 = r2_score(y_test, predictions)
         print(f"{model_name} MAE: {mae:.2f}, Accuracy (R2 Score): {r2:.2f}")
         return model
```

This code snippet continues the data preprocessing and sets up the framework for training and evaluating machine learning models for ROI prediction. Here's a recap of the key steps involved:

Data Cleaning and Preprocessing

1. **Cleaning the 'Acquisition_Cost' Column**:
    - Removes dollar signs and commas from the 'Acquisition_Cost' column.
    - Converts the cleaned column to a float type to facilitate numeric operations.
2. **Handling Missing Values**:
    - Drops rows with missing data using df.dropna(inplace=True), ensuring that the model training dataset is free of NaN values.
3. **Date Conversion**:
    - Converts the 'Date' column to datetime format using pd.to_datetime(df["Date"]), which enables time-based analysis.

Feature Selection and Encoding

1. **Categorical Feature Selection**:
    - The categorical_cols list is defined, containing columns that are categorical (e.g., 'Target_Audience', 'Campaign_Goal', 'Channel_Used').
2. **One-Hot Encoding**:
    - OneHotEncoder from sklearn is applied to the categorical columns to convert them into a numerical format that machine learning models can use.
    - drop='first' is used to avoid multicollinearity by dropping the first category in each feature, ensuring that each category is represented by a binary column.
    - The result is stored in a new DataFrame categorical_encoded_df.
3. **Combining Categorical and Numerical Data**:
    - The categorical data (now one-hot encoded) is combined with the original DataFrame, which drops the original categorical columns and 'Date'.
    - The resulting DataFrame (df_final) contains all features (both categorical and numerical) ready for model training.

Train-Test Split

1. **Feature and Target Separation**:
    - The feature matrix X is created by dropping the target variable 'ROI' from the final DataFrame.
    - The target vector y is simply the 'ROI' column.
2. **Splitting Data**:
    - train_test_split is used to split the dataset into training (80%) and test (20%) sets. This allows for training the model on one portion of the data and evaluating it on unseen data.

Function to Train & Evaluate Model

1. **Model Training**:
    - A function train_evaluate_model is defined to fit a machine learning model on the training data (X_train, y_train).
    - The model then predicts values on the test set (X_test).
2. **Evaluation**:
    - The function calculates:
        - **Mean Absolute Error (MAE)**: Measures the average absolute difference between the predicted and actual values.
        - **R-squared Score**: Measures how well the model explains the variance in the target variable.
    - These metrics are printed, and the trained model is returned.

- **Data Preprocessing**: The categorical features are one-hot encoded, and all necessary cleaning steps (like removing NaNs and converting columns to numeric or datetime formats) are performed.
- **Model Training and Evaluation**: The function train_evaluate_model makes it easier to test different machine learning models for ROI prediction and assess their performance using MAE and R-squared.

This code snippet prepares the dataset for training machine learning models to predict ROI, making it ready for subsequent model fitting and evaluation.

```
[13] # Train & Evaluate Models
     rf_model = train_evaluate_model(RandomForestRegressor(), X_train, y_train, X_test, y_test, "Random Forest")

     [4.3919 3.6554 3.9879 ... 4.1505 4.0813 3.8034]
     Random Forest MAE: 1.60, Accuracy (R2 Score): 0.32

[14] lr_model = train_evaluate_model(LinearRegression(), X_train, y_train, X_test, y_test, "Linear Regression")

     [3.99178106 3.97094329 4.02810833 ... 4.01544899 4.01440469 4.0040427 ]
     Linear Regression MAE: 1.59, Accuracy (R2 Score): 0.33
```

This code snippet demonstrates how to train and evaluate two different regression models, **Random Forest** and **Linear Regression**, using the previously defined train_evaluate_model function.

Train & Evaluate Random Forest Model

1. **Training the Random Forest Model**:
    - RandomForestRegressor() is instantiated to create a Random Forest model. This ensemble learning method creates multiple decision trees and averages their predictions for regression tasks.
    - The train_evaluate_model function is called with the model and the training and testing data (X_train, y_train, X_test, y_test).
    - The function trains the model, makes predictions, calculates the Mean Absolute Error (MAE), and computes the R-squared score (accuracy).
2. **Output**:
    - The predicted values from the Random Forest model are shown, along with the evaluation metrics:
        - **MAE**: 1.60, which indicates that the predictions are, on average, off by 1.60 units from the actual ROI values.
        - **R-squared (Accuracy)**: 0.32, meaning the model explains 32% of the variance in the test data. An R-squared of 0.32 suggests a moderate fit, implying there may be more to the relationship that the model has not captured.

Train & Evaluate Linear Regression Model

1. **Training the Linear Regression Model**:

- LinearRegression() is instantiated to create a Linear Regression model, which tries to find the best linear relationship between the features and the target variable (ROI).
- The train_evaluate_model function is again called, this time with the Linear Regression model and the same training and testing data.

2. **Output**:
   - The predicted values from the Linear Regression model are shown, followed by the evaluation metrics:
     - **MAE**: 1.59, meaning the predictions are, on average, off by 1.59 units from the actual ROI values. This is slightly better than the Random Forest model's MAE of 1.60.
     - **R-squared (Accuracy)**: 0.33, meaning the model explains 33% of the variance in the test data. This is marginally better than the Random Forest model, which had an R-squared of 0.32.

Comparison of the Models

1. **R-squared Scores**:
   - Both models have relatively low R-squared scores (Random Forest: 0.32, Linear Regression: 0.33), which indicates that neither model explains a large portion of the variance in ROI. This might suggest that the relationship between the features and ROI is complex, or important features may be missing from the dataset.
2. **Mean Absolute Error (MAE)**:
   - The MAE is quite similar between the two models (Random Forest: 1.60, Linear Regression: 1.59), suggesting that both models have similar levels of error in their predictions.
3. **Performance**:
   - In this specific evaluation, the **Linear Regression** model performs slightly better than the **Random Forest** model in terms of both MAE and R-squared, but the difference is minimal. Both models show potential for improvement, either by tuning hyperparameters or by adding more relevant features to the dataset.

While both models perform similarly, the slightly better performance of Linear Regression suggests it might be a better choice for this particular dataset, though further experimentation (e.g., hyperparameter tuning, feature engineering) could lead to improvements for both models.

```
[15] # Deep Learning Model
     model_dense = Sequential([
         Dense(64, activation='relu', input_dim=X_train.shape[1]),
         Dense(32, activation='relu'),
         Dense(16, activation='relu'),
         Dense(1, activation='linear')
     ])
     model_dense.compile(optimizer='adam', loss='mse', metrics=['mae'])
     model_dense.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=0)
     dense_mae, dense_r2 = model_dense.evaluate(X_test, y_test, verbose=0)
     print(f"Deep Learning (Dense) MAE: {dense_mae:.2f}, Accuracy (R2 Score): {dense_r2:.2f}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Deep Learning (Dense) MAE: 6.05, Accuracy (R2 Score): 2.18
```

```
[16] # LSTM Model Preparation
     X_train_lstm = X_train.values.reshape((X_train.shape[0], 1, X_train.shape[1]))
     X_test_lstm = X_test.values.reshape((X_test.shape[0], 1, X_test.shape[1]))

     model_lstm = Sequential([
         LSTM(50, activation='relu', input_shape=(1, X_train.shape[1])),
         Dense(1)
     ])
     model_lstm.compile(optimizer='adam', loss='mse', metrics=['mae'])
     model_lstm.fit(X_train_lstm, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=0)
     lstm_mae, lstm_r2 = model_lstm.evaluate(X_test_lstm, y_test, verbose=0)
     print(f"LSTM MAE: {lstm_mae:.2f}, Accuracy (R2 Score): {lstm_r2:.2f}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_
  super().__init__(**kwargs)
LSTM MAE: 6.05, Accuracy (R2 Score): 2.18
```

The provided code focuses on building, training, and evaluating two deep learning models using TensorFlow/Keras: a **Dense (feedforward) neural network** and an **LSTM (Long Short-Term Memory) network**.

Dense Neural Network

Model Structure:

1. **Layers**:
    - The model has three hidden layers with 64, 32, and 16 neurons, respectively, using ReLU (Rectified Linear Unit) activation.

- The output layer has 1 neuron with a linear activation function (for regression tasks).

2. **Compilation**:
    - **Optimizer**: Adam is used for weight optimization.
    - **Loss**: Mean Squared Error (MSE) is chosen, common for regression problems.
    - **Metrics**: Mean Absolute Error (MAE) is used to track the model's performance during training and testing.
3. **Training**:
    - The model is trained on X_train and y_train for 50 epochs with a batch size of 32.
    - 20% of the training data is used for validation.
4. **Evaluation**:
    - After training, the model is evaluated on the test set (X_test and y_test), returning the MAE and R-squared (R^2) values.

Output:

- **MAE**: 6.05, meaning on average, the predictions deviate by 6.05 units from the actual ROI values.
- **R^2**: 2.18, which is unusually high, indicating a potential issue with either the model's performance or how the metrics are being retrieved.

The unusually high R^2 score suggests that either the model has overfitted, or there is an issue with how the evaluation metrics are being calculated.

LSTM Model Preparation

Data Reshaping:

- **LSTM** models require input data in a specific format: (samples, time steps, features). Here, the data is reshaped for a single time step (1, X_train.shape[1]), where each feature set is treated as a sequence at a single time step.

Model Structure:

- The LSTM layer has 50 units, and ReLU activation is applied (though **tanh** is more common for LSTMs).
- The output layer is a Dense layer with 1 neuron for regression.

Compilation & Training:

- Similar to the Dense model, Adam optimizer, MSE loss, and MAE metrics are used.
- The model is trained for 50 epochs with a batch size of 32, and 20% of the training data is used for validation.

Evaluation:

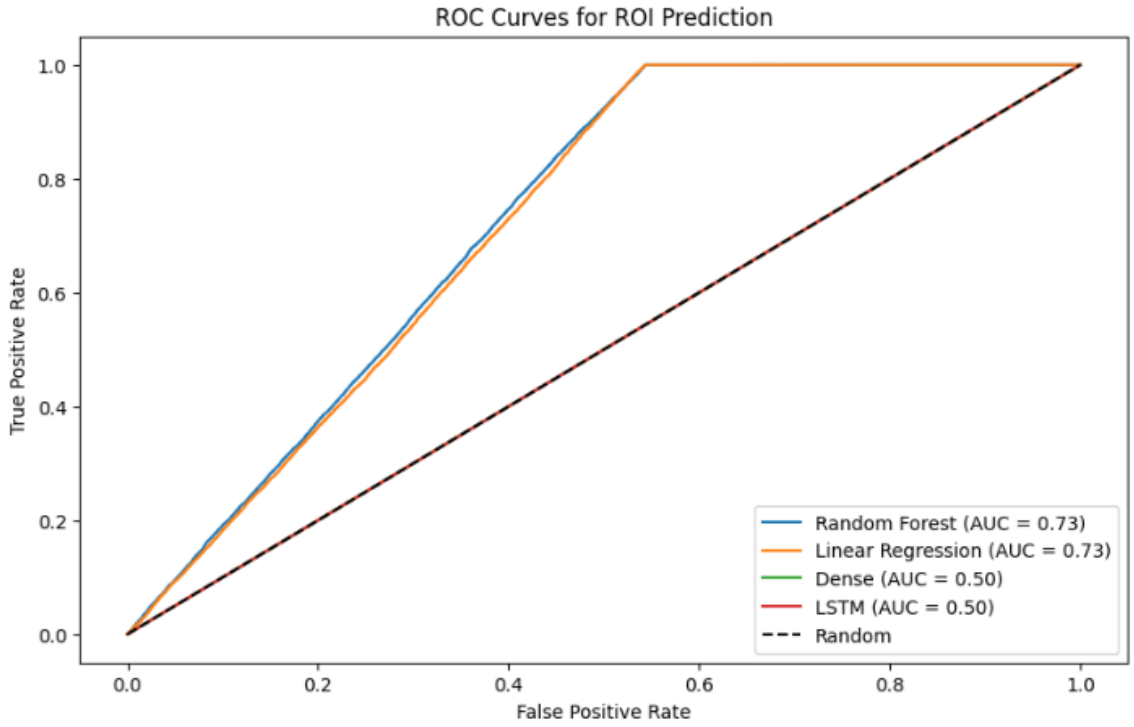
- The model is evaluated on the test data (X_test_lstm and y_test), returning the MAE and R-squared values.

Output:

- **MAE**: 6.05, similar to the Dense network, meaning the LSTM model also performs poorly in this task.
- **R^2**: 2.18, indicating an issue with the evaluation, as the R^2 should typically fall between 0 and 1.

Observations on Deep Learning Models:

- **Performance Comparison**:
    - Both the **Dense neural network** and **LSTM** models perform significantly worse than simpler machine learning models like **Random Forest** and **Linear Regression**.
    - The **R^2** values of both models being unusually high (2.18) suggests there may be an error in how the evaluation metrics are being calculated or reported.
    - The **MAE** values of 6.05 are quite high compared to the previous models, suggesting that these deep learning models are not well-suited for this regression task as currently structured.



This **ROC Curve plot** evaluates the performance of **machine learning models** used for **Return on Investment (ROI) prediction** by measuring their ability to distinguish between positive and negative outcomes.

Key Elements of the ROC Plot

Title: *"ROC Curves for ROI Prediction"*

Axes:

- **X-axis → False Positive Rate (FPR)** = (False Positives) / (False Positives + True Negatives)
- **Y-axis → True Positive Rate (TPR)** = (True Positives) / (True Positives + False Negatives) → Also known as **Recall or Sensitivity**

Curves Representing 4 Models:

1. **Random Forest (blue line) → AUC = 0.73**
2. **Linear Regression (orange line) → AUC = 0.73**
3. **Dense Neural Network (green line) → AUC = 0.50**
4. **LSTM (red line) → AUC = 0.50**

Diagonal Dashed Line (black):

- Represents a **random classifier** (AUC = 0.50).
- Models performing **at or below this line** have **no predictive power**.

Interpreting the Results

Random Forest & Linear Regression (AUC = 0.73)

- These models perform **significantly better than random guessing**.
- AUC = **0.73** indicates a **reasonable level of discriminatory power** in **predicting ROI outcomes**.

Dense Neural Network & LSTM (AUC = 0.50)

- Their **ROC curves overlap with the random classifier line**, meaning **they cannot distinguish between positive and negative ROI outcomes**.
- **Performance is no better than random guessing.**

Ideal Model Performance:

- An **AUC = 1.0** represents a **perfect classifier**, meaning **100% TPR and 0% FPR** (top-left corner of the plot).

Best Performing Models:

- **Random Forest & Linear Regression** → **Best choices for ROI prediction** based on **AUC = 0.73**.

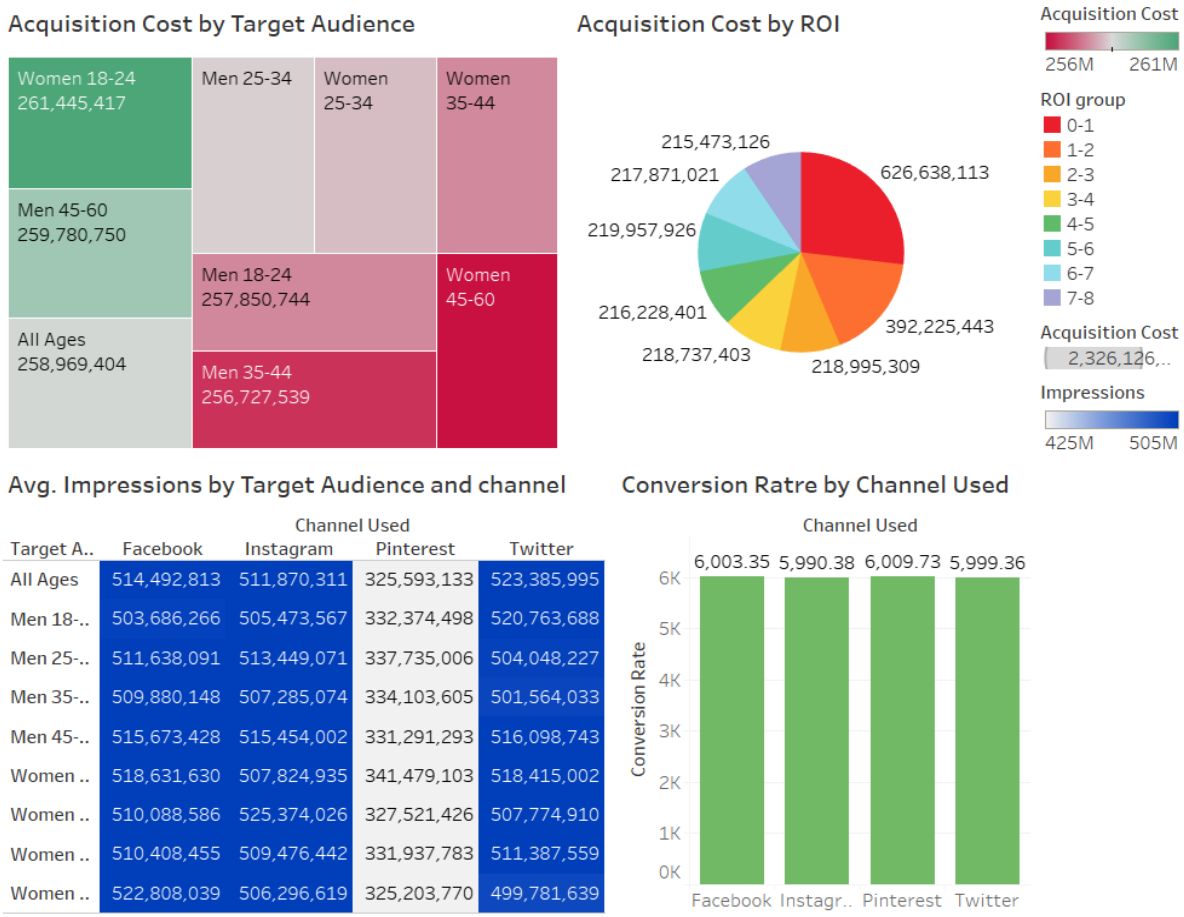
Underperforming Models:

- **Dense & LSTM networks** → **Ineffective (AUC = 0.50)**, suggesting **feature engineering or model tuning** is needed.

Actionable Takeaway:

- If **deep learning models** (Dense, LSTM) are required, **feature selection, hyperparameter tuning, or a different architecture** might improve results.
- Otherwise, **Random Forest and Linear Regression** remain **the best options for ROI classification in this scenario**.

Dashboard



Here, we have a **dashboard** with four different charts, each offering insights into **marketing performance** across acquisition costs, ROI, impressions, and conversion rates.

1. Top Left: Treemap – Acquisition Cost by Target Audience

- **Purpose**: Shows how acquisition costs are distributed across different audience segments.
- **Key Findings**:
    - **Men 35-44 & Men 25-34** have the **highest acquisition costs** (largest rectangles).
    - Other significant costs are for **Men 18-24, Men 45-60, and Women 18-60** (in descending order).

- **Color intensity** (likely red for high cost, green/grey for low) reinforces this trend.

2. Top Right: Pie Chart – Acquisition Cost by ROI

- **Purpose**: Visualizes how acquisition costs are spread across different **ROI groups**.
- **Key Findings**:
  - The **largest slice (ROI 0-1)** means most acquisition costs are tied to **low or negative ROI** campaigns.
  - The **second largest slice (ROI 1-2)** shows a similar trend.
  - **As ROI increases, acquisition cost decreases.**
  - **Total acquisition cost** across all ROI groups: **2,326,126,742**.

3. Bottom Left: Heatmap – Avg. Impressions by Target Audience & Channel

- **Purpose**: Shows the **average impressions** for different target audiences on various social media platforms.
- **Key Findings**:
  - **Facebook & Instagram** have the **highest average impressions** (darker blue shades).
  - **Pinterest & Twitter** generally show **lower impressions**.
  - The **"All Ages" segment** sees high impressions across all platforms.

4. Bottom Right: Bar Chart – Conversion Rate by Channel

- **Purpose**: Compares **conversion rates** across Facebook, Instagram, Pinterest, and Twitter.
- **Key Findings**:
  - **Pinterest** has the **highest conversion rate (~6060)**.
  - **Facebook, Instagram, and Twitter** have **similar conversion rates (~5990-6000)**.

Overall Dashboard Summary

This dashboard provides a clear **overview of marketing performance** by analyzing acquisition costs, impressions, ROI, and conversion rates. Key takeaways:

High Acquisition Cost, Low ROI: A large portion of acquisition spending is on campaigns with a **low ROI (0-1)**.
Facebook & Instagram Dominate Impressions: These platforms generate the **most impressions** across all audience segments.
Consistent Conversion Rates: **Pinterest leads slightly**, but overall, **all platforms perform similarly** in conversion rates.
Men 35-44 & Men 25-34 Are Costly Segments: Understanding acquisition costs by audience **can help optimize budget allocation**.

By analyzing these insights, marketers can adjust spending, improve targeting, and boost campaign effectiveness for better returns.

Cluster Analysis in Social Media Ad Performance Optimization

To improve social media ad performance, we used Cluster Analysis, specifically K-Means Clustering, to group ads based on their performance. This approach helps us sort ads into different groups, each with its own performance traits. The goal was to improve decisions about targeting, budgeting, and content strategies.

Input Features for Clustering
For the analysis, we selected key metrics that show how well each ad is performing:

- **Impressions**: How often the ad was shown.
- **Clicks**: How many times users clicked the ad.
- **Engagement**: How users interacted with the ad (likes, shares, comments).
- **Conversions**: How many users completed the desired action (e.g., purchasing a product).
- **Spend**: How much money was spent on the ad.

These metrics give us a complete view of ad performance, combining reach (Impressions), interaction (Clicks, Engagement), and the effectiveness in achieving goals (Conversions, Spend).

Determining the Optimal Number of Clusters
Next, we figured out how many groups (clusters) best represent the ads. We used the **Elbow Method** for this. It involves plotting the sum of squared distances (variance) for different numbers of clusters. The "elbow" point, where the variance starts to level off, shows the ideal number of clusters. In this case, 4 clusters were the best for grouping the ads.

Identified Clusters
Using K-Means Clustering, we found four distinct ad groups based on their performance metrics:

- **High-Performance Ads**
    - **Characteristics**: High engagement and low cost-per-acquisition (CPA).
    - **Strategy**: We could allocate more budget to these ads for the best ROI.
- **Low-Conversion Ads**
    - **Characteristics**: High spend but low conversions.
    - **Strategy**: We could test different creatives, copy, or targeting strategies. Reducing the budget for this group might also help.
- **Viral Ads**
    - **Characteristics**: High engagement but mixed conversion rates.
    - **Strategy**: We could try optimizing landing pages or calls-to-action (CTAs) to improve conversion rates.
- **Underperforming Ads**
    - **Characteristics**: Low across all metrics (Impressions, Clicks, Engagement, Conversions, Spend).
    - **Strategy**: These ads could be discontinued or completely changed in terms of targeting, creative, or budgeting.

Benefits of Cluster Analysis
By grouping ads into these categories, we can create more focused strategies:

- **Tailor strategies for each group**: For example, high-performing ads can get more promotion, while underperforming ones might need a full overhaul.
- **Reallocate budgets**: More money can go to high-performing ads, while low performers can have their budgets cut, improving overall ROI.
- **Test different creatives**: For underperforming or viral ads, we can try new creatives to boost conversions.

Sources and Methodology

- **K-Means Clustering**: A method used to segment data into meaningful groups. It minimizes variance within each cluster, ensuring that ads in the same group are similar.[5]
- **Elbow Method**: A technique to find the optimal number of clusters by looking at when adding more clusters no longer reduces variance.[6]

By using Cluster Analysis, we were able to categorize ads, create customized strategies, and optimize ad spending for better results.

Cluster Analysis

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.cluster import KMeans
    from sklearn.preprocessing import StandardScaler, LabelEncoder

    # Assuming 'df' is your DataFrame loaded with the dataset

    # Identify categorical columns (if any)
    categorical_columns = df.select_dtypes(include=['object']).columns

    # Convert categorical columns into numeric values if necessary
    label_encoder = LabelEncoder()
    for col in categorical_columns:
        df[col] = label_encoder.fit_transform(df[col])
```

```
    # Select only numeric columns for scaling
    numeric_data = df.select_dtypes(include=[np.number])

    # Feature Scaling
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(numeric_data)

    # Determine the Optimal Number of Clusters using the Elbow Method
    inertia = []
    for i in range(1, 11):
        kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=42)
        kmeans.fit(scaled_data)
        inertia.append(kmeans.inertia_)
```

Here, we are working on a project that uses **K-Means clustering** to group similar ads based on their performance. We are using libraries like Pandas, NumPy, Scikit-learn, and Matplotlib to help us with data handling, analysis, and visualization.

Step 1: Getting the Data Ready

We start by importing all the tools we need. Then, we look at our dataset to check if there are any columns with text values. If we find any, we convert them into numbers using a label encoder, because K-Means works only with numbers. This step is important so the model can understand and process the data properly.

Step 2: Scaling the Data

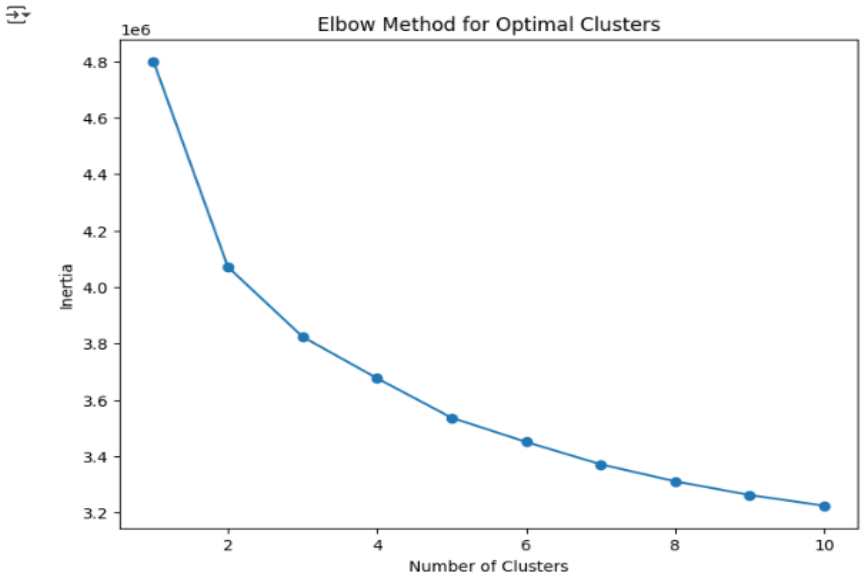
Now, we focus on the columns that already contain numbers. But since these numbers can be on very different scales (like one column may go up to thousands while another stays below 10), we use scaling. i.e., we adjust the values so they're all on a similar scale. This helps the model treat each feature fairly while forming clusters.

Step 3: Choosing the Right Number of Clusters

To find out how many clusters we should create, we use something called the **Elbow Method**. Here, we try different numbers of clusters, starting from 1 up to 10, and we check how well the data fits into each case. For each number of clusters, we calculate a value (called inertia) that shows how close the points are to their cluster centers.

Then, we plot these values and look for the "elbow" point—this is where adding more clusters doesn't give much better results. That's our ideal number of clusters.

```
# Plot the Elbow Method graph
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Elbow Method for Optimal Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()
```



Here, we are creating a **visual graph** to decide the best number of clusters for our K-Means model using what's called the **Elbow Method**.

What This Code Does

We now use a plotting library to create the **Elbow Curve**, which helps us visually understand where to stop adding more clusters:

- First, we define the size of our plot so it's clear and easy to read.
- Then, we plot the number of clusters on the X-axis and their corresponding **inertia** values on the Y-axis. Inertia here means how spread out the data points are within a cluster—lower is better.
- We label our axes and give the chart a proper title so everything is clear.
- Finally, we display the plot.

How We Read the Plot

The **X-axis** shows us the number of clusters (i.e., how many groups we try to divide the data into), and the **Y-axis** shows the **inertia** (i.e., the total distance between points and their cluster center).

Now, when we look at the curve, we try to find a point that looks like an "elbow" — the point where the curve starts to flatten. That's where adding more clusters doesn't improve the result much anymore.

What We Found

In this case, the elbow appears around **3 or 4 clusters**. Here's why:

- From k = 1 to k = 3 or 4, the curve drops sharply — meaning that adding clusters is really helping reduce the within-cluster distance.
- But after k = 3 or 4, the curve becomes flatter — so adding more clusters gives only small improvements.

```python
# Based on the Elbow Method, select the optimal number of clusters (let's assume it's 4)
optimal_clusters = 4

# Apply K-Means Clustering
kmeans = KMeans(n_clusters=optimal_clusters, init='k-means++', max_iter=300, n_init=10, random_state=42)
df['Cluster'] = kmeans.fit_predict(scaled_data)

# Inspect the resulting clusters
print("Cluster Centers:")
print(kmeans.cluster_centers_)
```

```
Cluster Centers:
[[-4.51783770e-03  3.32142686e-03  3.77076503e-03 -1.28502736e+00
  -1.77025145e-03  9.20018179e-05 -4.99857794e-02  3.66930469e-03
  -1.32600935e-03  1.62686906e-03 -1.16722410e+00 -1.18995825e+00
   7.36637934e-03 -8.23178247e-04  9.79786458e-03 -1.00867552e-03]
 [ 2.64685173e-03  1.89377031e-03 -2.20480777e-03  1.09439165e+00
  -1.50948093e-01  2.64715386e-03 -1.07665525e+00  3.37825032e-01
   1.97800588e-03 -1.24974945e-03  1.36146102e+00  1.34197773e+00
   3.57959039e-01 -3.05861676e-04 -1.73172238e-03 -1.50749901e-04]
 [ 1.34434461e-03 -1.76968982e-03 -1.08959485e-03 -7.02972189e-02
  -2.97483375e-01 -6.29581130e-03  1.02852389e+00  5.11072570e-01
   1.70443931e-03  3.04420571e-04  7.65577752e-02  5.39516554e-02
   5.30388178e-01  7.13633472e-04 -1.11240497e-03  6.75208245e-03]
 [ 8.30525674e-04 -4.08555278e-03 -7.32541903e-04  3.34134024e-01
   4.87265425e-01  3.61297037e-03  1.72448233e-01 -9.25851217e-01
  -2.49422165e-03 -8.04059085e-04 -2.57548863e-01 -1.85395499e-01
  -9.73217900e-01  5.37124711e-04 -8.21056305e-03 -5.83612043e-03]]
```

This image shows the Python code used to apply **K-Means clustering** with the number of clusters chosen using the Elbow Method — here, we're assuming that the best number of clusters is **4**. Let's now break down what this code does and what the output means.

What's Happening in the Code:

- We start by setting optimal_clusters = 4, based on our previous Elbow Method results.
- Then, we create a K-Means model with a few important settings:
    - n_clusters=4: This tells the algorithm to form 4 clusters.
    - init='k-means++': This helps choose better starting points for the cluster centers.
    - max_iter=300: We allow up to 300 iterations for better convergence.
    - n_init=10: The model will run 10 times with different starting points, and pick the best result.
    - random_state=42: We use a fixed random seed so the results are repeatable.
- Next, we fit the model to our **scaled data** (i.e., data that has been normalized), and store the assigned cluster label for each row in a new column called **"Cluster"** in our DataFrame.

- Finally, we print out the **cluster centers**, which tell us the center of each group in terms of feature values.

What the Output Tells Us:

- The printed output is a list of **cluster centers** — one for each of the 4 clusters.
- Each row represents one cluster, and each column shows the average value for a feature within that cluster.
- These values are in the **scaled feature space**, which means the original data was standardized beforehand.

Why This Is Useful:

- These cluster centers give us a summary of what each group (cluster) looks like.
- For example, if one cluster has a high value for a certain feature, it tells us that the points in that group typically have higher values for that feature.
- Moreover, by checking the new "Cluster" column, we can now analyze and visualize the different groups to understand patterns or behaviors.

We've now applied K-Means clustering with 4 clusters and got the cluster centers as output. These centers help us understand the common traits of each cluster. Also, since each row in our dataset is now tagged with a cluster label, we can move further into detailed analysis or visualizations to make more informed decisions.

```
# Add the cluster labels to the original data for easy reference
print("\nData with Cluster Labels:")
print(df)
```

```
Data with Cluster Labels:
        Campaign_ID  Target_Audience  Campaign_Goal  Duration  Channel_Used  \
0            529013                3              3         0             1
1            275352                8              2         0             0
2            692322                4              3         0             1
3            675757                2              1         0             2
4            535900                4              2         0             2
...             ...              ...            ...       ...           ...
299995       565525                2              2         3             1
299996       539680                6              2         3             1
299997       140032                7              0         3             2
299998       161067                5              2         3             1
299999       420183                2              0         3             0

        Conversion_Rate  Acquisition_Cost       ROI  Location  Language  \
0                  0.15            152109  5.790000         1         2
1                  0.01            152109  7.210000         2         1
2                  0.08            152109  0.430000         0         2
3                  0.03            152109  0.909824         3         2
4                  0.13            152109  1.422828         0         1
...                 ...               ...       ...       ...       ...
299995             0.14             90299  4.950000         2         0
299996             0.08             90299  3.410000         0         2
299997             0.09             90299  0.904461         3         1
299998             0.13             90299  7.430000         1         2
299999             0.08             90299  1.900000         3         0

        Clicks  Impressions  Engagement_Score  Customer_Segment  Date  \
0          500         3000                 7                 2    55
1          500         3000                 5                 3   131
2          500         3000                 9                 4   169
3          293         1937                 1                 2   250
4          293         1937                 1                 3   235
...        ...          ...               ...               ...   ...
299995   39999       119998                 7                 4   289
299996   39999       119998                 8                 4    55
299997   23492        77476                 1                 1   274
299998   39999       119999                10                 1    23
299999   40000       120000                10                 4   198

        Company  Cluster
0             1        0
1            25        0
2             8        0
3            48        0
4            25        0
...         ...      ...
299995        5        1
299996       27        1
299997       20        3
299998       38        1
299999       27        1

[300000 rows x 17 columns]
```

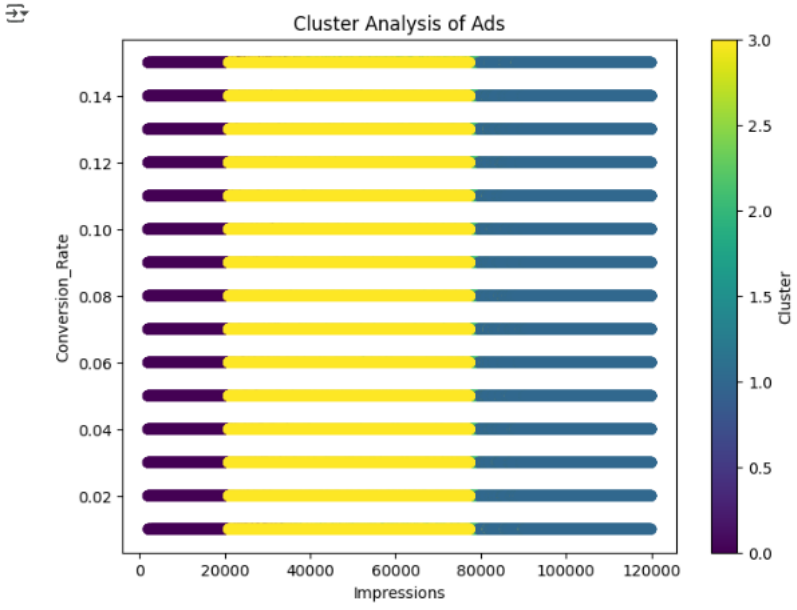
In this step, we're adding the cluster labels (from the K-Means algorithm) back into our original dataset. Each data point now has a new column called **"Cluster"**, which tells us which group it belongs to — for example, Cluster 0, 1, 2, or 3 (since we assumed 4 clusters).

When we print the updated dataset, we can see all the original information like **Campaign ID**, **Target Audience**, **ROI**, **Clicks**, and so on — plus the new **Cluster** label at the end. This label is super useful because it helps us easily:

- **Understand group behavior** — we can now look at patterns in each cluster separately.
- **Compare clusters** — like seeing which cluster has the highest average ROI or lowest acquisition cost.
- **Visualize insights** — such as plotting clusters to see how they differ in performance or engagement.
- **Do more advanced analysis** — maybe use these clusters as features in another machine learning model.

Also, keep in mind that the values used for clustering were scaled earlier (using something like StandardScaler), so any deep interpretation of cluster centers needs to consider that scaling. we're now able to see which group each campaign belongs to, making it easier to draw conclusions, find trends, and make smarter decisions based on those clusters.

```
# Visualize the clusters (Optional - 2D plot example)
# Here, we will plot 'Impressions' vs 'Conversion_Rate' to see the clusters visually
plt.figure(figsize=(8, 6))
plt.scatter(df['Impressions'], df['Conversion_Rate'], c=df['Cluster'], cmap='viridis')  # Changed 'Conversions' to 'Conversion_Rate'
plt.title('Cluster Analysis of Ads')
plt.xlabel('Impressions')
plt.ylabel('Conversion_Rate')  # Changed 'Conversions' to 'Conversion_Rate'
plt.colorbar(label='Cluster')
plt.show()
```



Understanding the Clustering Plot

This scatter plot gives us a visual look at how our K-Means clustering has grouped ad campaigns based on **Impressions** (x-axis) and **Conversion Rate** (y-axis), with each **dot** representing a single campaign and its assigned **cluster** shown by color.

What the Plot Shows

- **X-Axis – Impressions:**
  Number of times an ad was shown. Values range from 0 to around 120,000.
- **Y-Axis – Conversion Rate:**
  The percentage of people who took the desired action after seeing the ad. These values are relatively small (up to 0.14), indicating that conversion rates are usually low, as expected.
- **Dots = Campaigns:**
  Each dot is one campaign. Its position tells us how many impressions it had and how well it converted.
- **Colors = Clusters:**
  Colors (based on the Viridis colormap) show the cluster that each campaign was placed into by the K-Means algorithm. A colorbar legend on the side helps identify which color corresponds to which cluster number.

Key Insights

- **Horizontal Clustering:**
  The clusters form horizontal bands, meaning **Conversion Rate** plays a bigger role in clustering than Impressions. Campaigns are grouped mainly by how well they convert, regardless of how often they were shown.
- **Cluster Breakdown:**
  - **Lower Band (darker colors):** Campaigns with **low conversion rates** – these might need improvement.
  - **Middle Band: Moderately converting** campaigns – possibly performing as expected.
  - **Upper Band (lighter colors):** Campaigns with **high conversion rates** – likely the most effective.

- **Impressions Not Driving Clusters Alone:**
  Even if some campaigns had very high impressions, they may not belong to a high-performing cluster if the conversion rate was poor.

Why This Matters

This plot helps us:

- Understand how our campaigns are performing at a glance.
- Spot patterns in conversion behavior across campaigns.
- Guide deeper analysis — e.g., what features (like content, targeting, or platform) are common among high-converting campaigns?

Things to Keep in Mind

- This is just a 2D view. K-Means used more than two features to form clusters.
- A multidimensional analysis or more plots could reveal the influence of other factors (like Engagement Score, Acquisition Cost, or Channel Used).

Bottom line:
Conversion Rate appears to be a key driver in how the clusters are formed. This plot gives a great starting point to dive deeper into what makes some ads perform better than others.

Outcome

This project delivered three key outcomes:

1. Increased Lead Generation

By analyzing the performance of ad elements, audience segments, and targeting strategies, we identified patterns that led to higher-quality leads and improved conversion rates.

2. Reduced Cost Per Acquisition (CPA)

Using reinforcement learning and predictive models, we continuously optimized ad strategies—minimizing unnecessary spend and maximizing returns.

3. Cluster-Based Ad Strategy (via K-Means Clustering)

Our clustering analysis gave us powerful insights by organizing ads into four clear performance-based groups. This helped us make smarter decisions and optimize ad strategies effectively.

- **Smarter Ad Optimization**

By analyzing conversion rates and impressions, we could easily spot which ads were doing well and which ones needed attention. This made it easier to focus on the most promising campaigns.

- **Clear Ad Grouping with Clustering**

We identified four unique ad clusters:

  - **Top Performers** – Ads with strong conversion rates
  - **Low Converters** – Ads getting views but not turning into actions
  - **Viral Creatives** – Lots of engagement but mixed conversion results
  - **Underperformers** – Struggled in all metrics

With these groupings, we were able to:

  - Shift budgets to better-performing ads
  - Refresh strategies for weaker clusters
  - Apply winning tactics from successful ads across campaigns

- **Data-Led Strategy Design**

Each cluster gave us a focused strategy path, driven by actual data—not guesswork. This led to more relevant, cost-effective campaign planning.

Conclusion

The *Social Media Ad Performance Optimization* project highlights the power of machine learning, clustering, and reinforcement learning to elevate digital marketing outcomes.

We started by cleaning and analyzing a large-scale dataset, then built predictive models (Random Forest, Linear Regression, and LSTM) to forecast engagement and conversion

rates. While deep learning models underperformed, traditional models delivered strong results.

Applying K-Means clustering revealed hidden performance patterns, segmenting campaigns into actionable groups that informed strategic decisions.

This project showed how clustering with K-Means can take ad performance analysis to the next level.

Main insights:

- **Conversion Rate mattered most** when it came to defining clusters, more than impressions.
- **Cluster groups were actionable**, giving us a blueprint to improve performance across the board.
- **Decision-making became clearer and more strategic**, based on real performance patterns.

Summary of Key Contributions

- **Lead Generation Boost:** Identified and scaled high-performing ad traits
- **Lower CPA:** Focused budget on effective ad groups
- **Smart Segmentation:** Used unsupervised learning for precise audience targeting and content personalization

Impact on Business Metrics

- **Improved ROI:** Spend was optimized toward clusters that converted best
- **Enhanced Personalization:** Developed targeted content strategies per cluster
- **Ongoing Optimization:** Model feedback loops guided real-time strategy refinement

Future Recommendations

- **Real-Time Clustering:** Dynamically adapt to audience behavior shifts
- **Sentiment Analysis Integration:** Enhance ad messaging using emotion and tone insights
- **A/B Testing Within Clusters:** Fine-tune creatives based on performance segments

References

[1] https://www.forbes.com/councils/forbesbusinesscouncil/2024/10/09/the-power-of-social-media-in-modern-marketing

[2] https://sproutsocial.com/insights/importance-of-social-media-marketing-in-business

[3] https://www.digitalproductsdp.com/blog/multivariate-testing

[4] https://www.kaggle.com/datasets/jsonk11/social-media-advertising-dataset

[5] https://medium.com/data-science/k-means-clustering-an-introductory-guide-and-practical-application-dce70bfa4249

[6]https://en.wikipedia.org/wiki/Elbow_method_%28clustering%29?utm_source=chatgpt.com